Centres étrangers – 2025 – sujet2 - Correction

Exercice 3 (8 points)

Question 1

Un exemple de chemin non prolongeable : 3 → 9 → 1 → 2 → 6 → 4 → 8.

- Commentaire : Il est non prolongeable car le dernier nombre (8) a pour diviseurs 1, 2 et 4, tous déjà présents dans le chemin, et n'a pas d'autre multiple ≤ 9.

Question 2

Les valeurs prises par la variable `valeur` sont : 1, 2, 3, 4, 5, 6, 7, 8, 9.

- Commentaire : La boucle `for valeur in range(1, n+1)` parcourt les entiers de 1 à n inclus.

Question 3

La valeur de `jeu_9[0].valeur` est 1, car c'est la première valeur ajoutée à la liste `jeu`.

Question 4

Ce test vérifie si `sommet.valeur` est un diviseur de `self.valeur`. Par exemple, 6 % 3 == 0 signifie que 3 est un diviseur de 6.

Question 5

`jeu_9[5]` représente le sommet de valeur 6. Ses diviseurs parmi 1 à 9 sont : 1, 2, 3.

Donc `jeu_9[5].diviseurs` contiendra les objets `Sommet` de valeurs 1, 2 et 3.

Question 6

```
jeu[valeur-1].relier_diviseurs(jeu)
```

- Commentaire : On relie chaque sommet à ses diviseurs en appelant la méthode correspondante.

Question 7

```
return [sommet.valeur for sommet in self.diviseurs]
```

- Commentaire : On extrait la valeur de chaque sommet dans la liste des diviseurs.

Question 8

```
l_div_3 = jeu_9[2].lister_diviseurs()
l_mult_3 = jeu_9[2].lister_multiples()
assert 1 in l_div_3
assert 6 in l_mult_3
assert 9 in l_mult_3
```

- Commentaire : `jeu_9[2]` correspond au nombre 3, ses diviseurs sont [1], et ses multiples [6, 9].

Question 9

Cette ligne vérifie que la file n'est pas vide avant de retirer un élément. Si la file est vide, l'opération de retrait n'a pas de sens et provoquerait une erreur. L'`assert` permet donc de stopper l'exécution avec un message d'erreur dans ce cas.

Question 10

```
return len(self.donnees) - self.decalage
```

- Commentaire : La taille de la file correspond au nombre d'éléments non encore défilés.

Question 11

```
f = File()
f.enfiler(1)
f.enfiler(2)
f.enfiler(3)
assert f.taille() == 3
assert f.defiler() == 1
assert f.defiler() == 2
assert f.defiler() == 3
assert f.est_vide()
```

Ce test vérifie que les éléments sont défilés dans le bon ordre et que la file est bien vide à la fin.

Question 12

```
    L_chemins = []
    f = File()
    for s in jeu:
        f.enfiler([s])
    while not f.est_vide():
        chemin = f.defiler()
        dernier = chemin[-1]
        voisins = dernier.diviseurs + dernier.multiples
        prolonge = False
        for v in voisins:
            if v not in chemin:
                f.enfiler(chemin + [v])
                prolonge = True
        if not prolonge:
            L_chemins.append(chemin)
    return L_chemins
```

Cette fonction explore tous les chemins non prolongeables possibles dans le graphe.

Question 13

```
def valeurs_chemin(chemin):
    return [s.valeur for s in chemin]
```

Cette fonction transforme en compréhension une liste de sommets en liste de leurs valeurs entières. On pouvait aussi utiliser une `boucle for`.

Question 14

```
chemins = []
for chemin in rechercher_chemins(jeu_9):
    chemins.append(valeurs_chemin(chemin))
```

On utilise la fonction précédente pour convertir les chemins en listes d'entiers.

Question 15

```
def extraire_plus_longs_chemins(L_chemins):
    max_len = max(len(c) for c in L_chemins)
    return [c for c in L_chemins if len(c) == max_len]
```

Cette fonction extrait tous les chemins de longueur maximale.

Question 16

Non. L'algorithme explore tous les chemins possibles en profondeur, ce qui provoque une explosion combinatoire du nombre de chemins à explorer. Le temps d'exécution et la mémoire utilisée deviennent très grands pour des $n > 14$. Le programme n'est donc pas adapté à des tailles de jeu très grandes.